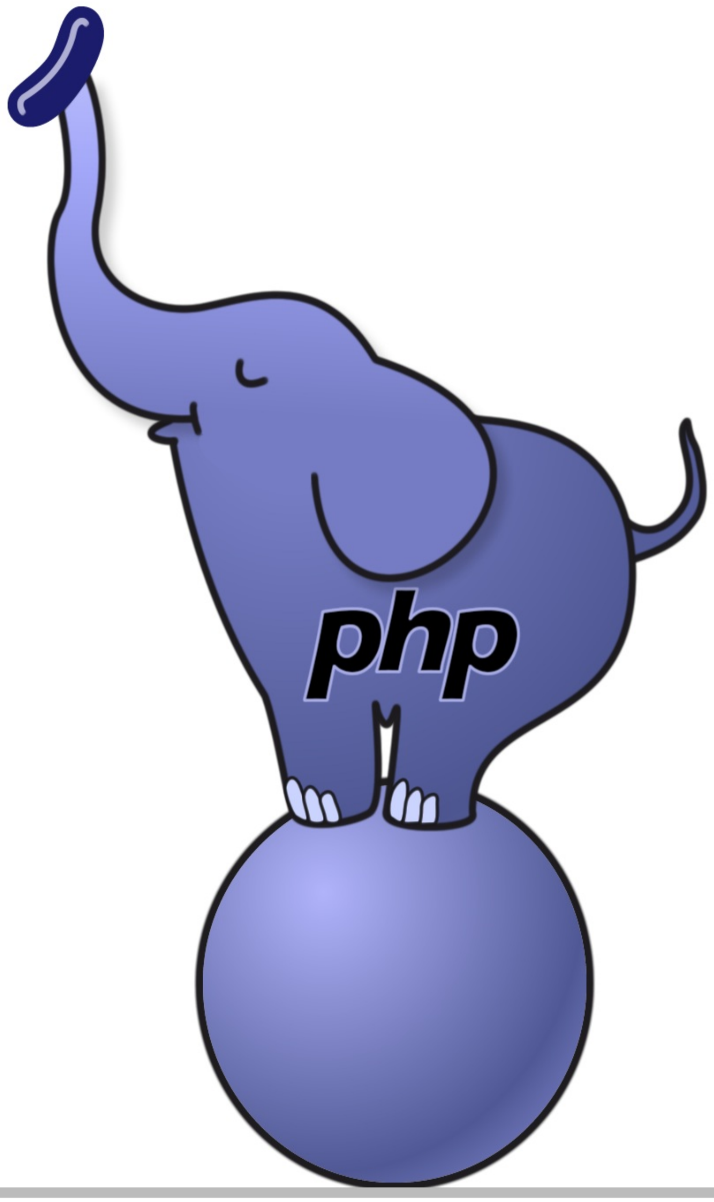
PHP

扩展开发入门



马秉尧 著

目錄

1. 前言
2. PHP 扩展开发环境和工具
    i. 配置开发环境
    ii. 选取开发工具
3. 如何来实现一个完整的 PHP 扩展
    i. 扩展的基本骨架
    ii. 将扩展项目导入 Netbeans
    iii. PHP 扩展的头文件
    iv. PHP 扩展的主文件

PHP 扩展开发入门

关于 PHP 扩展开发的书，在市面上是否存在，我是不知道的。至少我还没见过中文版的。不过网上有几本国内高人们写的，或者翻译的国外高人写的电子书，其中我知道的，一本是：深入理解PHP内核(Thinking In PHP Internals)，另一本是：PHP扩展开发及内核应用，其中后一本好像还有其它一个或几个版本。另外还有著名的国内 PHP 大神鸟哥的博客，Swoole 开发大神韩天峰（Rango）的博客，黑夜路人的开源世界上也有一些关于 PHP 扩展开发的文章。我就是从读这些书和文章入门的。所以在这里我要首先感谢以上各位大神们能够提供这些十分优秀且珍贵的资源。

尽管已经有了上面这些资源，但是要开始编写扩展，感觉还是很难。一方面是上面这些书或文章的内容都比较深，像我这样的初学者，反正有一半内容是看不懂或者看不下去的（当然看不下去还有个原因是因为有些内容还用不到），另一方面是上面这些书或文章的内容有一部分是比较陈旧的，比如还有不少的篇幅是来介绍关于 PHP 4 扩展编写的。但是关于主流的 PHP 5.3 以上以及即将到来的 PHP 7 的内容，介绍的就比较少了（也不是说完全没有哈）。另外，关于实际开发中，会遇到的一些细枝末节的问题，介绍的也比较少，一方面可能大神们觉得这些内容是不值一提的，没有必要写，另一方面可能大神们已经对扩展的编写轻车熟路，所以也遇不到这些基础的小问题。

所以作为一个刚刚编写了两个小扩展，算是刚刚从 PHP 扩展编写的坑里爬出来的我，在这个时候做一下总结，我觉得还是有必要的。至少，或者说但愿，这些总结能够让我以后编写扩展的时候，尽量不再掉进这些坑里，或者掉进去之后能够更快的爬上来，而不至于再抓狂好几天。

大神们写的扩展，有很多都是关于操作 PHP 字节码或者对 PHP 本身开刀做手术的。编写这种扩展需要的知识很多，也很深。我是不懂的，所以我就不写了。有兴趣的同学可以去读读上面介绍的那些书和文章，或者读一读 PHP 源码，也许能找到你需要的东西。

本书[1]的主要内容是以我所编写的两个扩展为例子，介绍 PHP 扩展开发中所遇到的一些实际问题以及解决方法，希望本书能够在帮助自己的同时，也能为想要学习 PHP 扩展开发的初学者们提供一些方便。

本书要讨论的两个 PHP 扩展一个是 xxtea，另一个是 hprose。

其中 xxtea 扩展是在多年之前就写好的，不过当时写的并不规范，直到最近发布到 pecl 上之后，才了解到编写一个规范的扩展需要注意哪些东西。我打算通过剖析这个扩展，来展示一下编写一个简单规范的扩展的流程。

而 hprose 扩展是多年之前就打算写，但直到最近才完成的一个扩展。这个扩展同样没有用到高深的知识，甚至连输入输出都没有涉及到，但是对于基础的东西基本都涉及到了。比如各种基本类型的操作，zval 变量的创建和销毁，HashTable，数组，类，接口，调用用户函数，方法和闭包，创建用户自定义类的对象，抛出异常，捕获异常等等。通过对这个扩展的剖析，我们基本上对 PHP 扩展开发中所遇到的基础问题可以有一个比较全面的了解。

这两个扩展既支持 PHP 5，也支持 PHP 7。所以本书也会把 PHP 5 和 PHP 7 扩展开发中的异同点，以及如何编写可以同时运行在这两个大版本上的扩展作为重点来讲解。

当然因为本人水平有限，加上 PHP 7 尚未发布，对一些内容的介绍上肯定会存在错误或漏洞，希望读者在谅解的同时，能够给予批评或指正。

本书暂未想好该如何规划结构，所以想到哪儿就先写到哪儿，对于写过的东西也会反反复复的修改，就像画画和编写程序一样。包括您已经看过的内容，也包括这一段。您不必为看过的东西可能在未来被删除而感到奇怪，如果您想找回它们，到 github 的历史记录里去翻查即可，这也是我为何要借助 github 来编写此书，而不是写在我的个人博客上的一个原因。

好了，废话不多说，因为已够多。下面就进入正文吧。

---

[1] 暂且称之为书吧，尽管才刚刚开始写，能写多少还不知道呢。

PHP 扩展开发环境和工具

工欲善其事，必先利其器

在开发 PHP 扩展之前，我们先得配置好开发环境，选取好开发工具，才好开始工作。

配置开发环境

如果我们开发扩展不仅仅是为了自己用的话，那么我们就应该考虑让它支持尽可能多的 PHP 版本。但是不同版本的 PHP 在 API 上会一些不同。尤其是 PHP 5 和 PHP 7 之间的差别更大。所以我们最好是搭建一个安装有多版本 PHP 的开发环境。

我使用的操作系统是 Mac OS X，并在上面安装了一个 ubuntu 的虚拟机，又在 ubuntu 里用 dockor 安装了一个 centos。

我使用的 Mac OS X 是 10.10.2 版本，可以使用内置的 PHP，但是要开发扩展，还需要安装一下 XCode。Linux 下首先要保证安装了 gcc，make 这些基本开发工具。然后再用 `apt-get install php-dev`（debian/ubuntu 下）或 `yum install php-devel`（redhat/centos 下）安装 PHP 扩展开发环境就可以了。

但是仅有这些内置的开发环境还是不够用的。要是自己手动下载 PHP 源码来编译安装各个不同版本的开发环境呢，倒是也可以。但我是个懒人，所以我找了两个更方便的工具来安装各个不同版本的 PHP 开发环境，并且可以随意切换。它们分别是 phpenv 和 php-build，至于使用方法呢，这两个工具的使用说明上都写的很清楚了，我就不多说了。

这三个系统下都可以使用这两个工具来编译安装 PHP 开发环境。至于 Windows 下是否支持，我就不知道了，因为我还没有尝试过在 Windows 下开发。所以 Windows 下如何配置开发环境，我就不介绍了。

选取开发工具

牛 A 和牛 C 之间的程序员通常喜欢使用 VIM 或 Emacs 加上一堆插件来做开发，但是对我这种连 Eclipse 都用不惯的菜鸟级的程序员来说，我更喜欢 Netbeans，只因为它配置简单，使用方便。毕竟我不会直接在服务器上用命令行做开发，而且我的电脑也没有差到只能跑字符界面系统的程度，所以就怎么简单怎么来咯。其实开发工具除了上面这几个以外还有很多，选你自己喜欢的就行了。

调试工具呢，在 Linux 下用 gdb， Mac OS X 下用 lldb 就可以了，它俩的用法差不多。通常就是你的测试程序跑崩了之后，会生成一个 core 文件，之后用 `gdb /path/to/php core` 打开 core 文件，用 bt 命令查看是从哪行崩溃的就可以了。这里的 `/path/to/php` 你得替换成实际的 php 命令的路径。如果你的程序跑崩了却没有产生 core 文件，你可以试试执行 `ulimit -c unlimited` 这条命令，然后再执行你的程序，你一般就会找到那个 core 文件了，如果还找不到，那就到 /cores，/tmp 或 /var/tmp 下看看有没有，在这些目录下的 core 文件通常会带一个数字的后缀，你只要知道就是那个东西就行了。你也可以先用 `gdb /path/to/php` 来启动 gdb，然后用 `run test.php` 来运行你的测试程序，如果执行崩了，你会直接看到崩溃信息。这个调试工具的功能当然不是只有这么多，不过我就会这么多，但感觉已经足够用了。如果你想要了解更多，可以去查这方面的资料，反正这种资料是不缺的。

不过有的时候，你为扩展写的测试程序虽然跑崩了，但是用 gdb 却很难定位出错的位置；或者你的测试程序没有跑崩，但是运行的结果却出现了一些莫名其妙的情况；又或者你的程序既没崩溃，结果也很正常，但是却存在内存泄漏（memory leak）[1]。这个时候，你就需要另外一个工具 valgrind 来定位这些错误了。这个工具的使用也很简单。下面是我曾经用过一条命令行：

```
USE_ZEND_ALLOC=0 valgrind /home/andot/.phpenv/versions/master/bin/php  -n -c '/mnt/hgfs/Work/Git/hprose-pecl/tmp-php.in
```

也许你会惊呼，怎么这条命令这么长，这让我怎么输入啊，啊，啊……

其实我也不是一个字符一个字符敲进去的，而是复制上去的。如果你的测试程序没有跑通过，在测试目录下，就会生成一个跟测试文件同名，但扩展名是 sh 的文件。你打开这个文件，把里面的命令行复制出来，贴到 `USE_ZEND_ALLOC=0 valgrind` 的后面就可以了。也许你可能会说，那我直接在后面跟脚本的名字不就行了。你可以试一下，虽然也可以出结果，但你会发现结果是不一样的。

如果你的程序中有内存泄漏，或者有重复的内存释放，或者释放了不该释放的值。那么你就可以看到这些错误的详细信息了。

不过 valgrind 这个工具似乎对 Mac OS X 10.10 支持的不是很好，至少还没有官方版本，通过 homebrew 也安装不了。这也是我为啥还要搞个 Linux 虚拟机的原因之一[2]。

开发环境和工具先介绍到这儿，以后要是有新发现再补充。接下来我们就进入实战阶段吧。

---

[1] 如果你在编译 PHP 开发环境时开启了 debug 模式，而你的扩展又存在内存泄漏，有时候你可以在运行结果中看到内存泄漏的提示。

[2] 另一个原因是在不同系统下编写扩展时，本身也有一些差别，所以需要安装多个系统来做测试。

如何来实现一个完整的 PHP 扩展

在编写我们的扩展之前，我们先来下载一个现成的扩展试试看能不能编译通过。我们就以 xxtea 扩展为例吧。这里我们采用的安装方法不是：

```
pecl install xxtea
```

尽管这是 pecl 扩展的标准安装方式。但我们的目的并不是安装使用它，而是通过它来了解一下如何编写自己的扩展。所以我们使用源码方式来安装：

```
git clone https://github.com/xxtea/xxtea-pecl
cd xxtea-pecl
phpize
./configure
make
make test
```

如果这个过程中没有出现任何错误，那么说明你的开发环境已经配置好了。

扩展的基本骨架

如果你是通过源码编译方式安装的 PHP，并且还保留了你安装时的 PHP 源码。那么你在源码的 ext 目录下可以找到一个叫 ext_skel 的程序。它的功能正如它的名字，就是用来生成一个扩展的基本骨架的。

例如，假设我们还没有开始写 xxtea 这个扩展，并且我们已经在 PHP 源码的 ext 目录下了，下面我们执行：

```
./ext_skel --extname=xxtea
```

我们会看到这样的输出内容：

```
Creating directory xxtea
Creating basic files: config.m4 config.w32 .svnignore xxtea.c php_xxtea.h CREDITS EXPERIMENTAL tests/001.phpt xxtea.php

To use your new extension, you will have to execute the following steps:

1.  $ cd ..
2.  $ vi ext/xxtea/config.m4
3.  $ ./buildconf
4.  $ ./configure --[with|enable]-xxtea
5.  $ make
6.  $ ./php -f ext/xxtea/xxtea.php
7.  $ vi ext/xxtea/xxtea.c
8.  $ make

Repeat steps 3-6 until you are satisfied with ext/xxtea/config.m4 and
step 6 confirms that your module is compiled into PHP. Then, start writing
code and repeat the last two steps as often as necessary.
```

通过这些信息，我们可以了解到，它帮我们在 ext 目录下，创建了一个 xxtea 的目录，并且在其中创建了一堆文件，这些文件就是一个 PHP 扩展的基本骨架。再后面它给出了一个编辑扩展的基本步骤，大致上呢，就这么多东西。

关于 ext_skel 以及它生成 config.m4 和 config.w32 这两个文件的介绍，读者可以参考官方的 PHP 手册 — PHP 核心：骇客指南 — PHP 5 构建系统，其它几个文件介绍参见：PHP 手册 — PHP 核心：骇客指南 — 扩展的结构，我这里就不重复了。

虽然上面说了这么多关于 ext_skel 的内容，但在实际开发扩展时，我们可能压根就不需要 ext_skel 这个程序。

为什么不需要 ext_skel 呢？首先一个原因是，它生成的这个骨架太过简单，在这个骨架的基础上修改又比较麻烦，还不如找一个现成的扩展，复制一份，修改一下来的方便。第二个原因是，不同版本的 PHP 生成的这个扩展骨架还是不一样的，如果你要编写的是跨版本的扩展，用这个东西也不太合适。它生成的东西，除了文件名以外，大部分东西你可能是需要重写的。

所以，即使你没有保留源码也无所谓，你不是通过编译源码方式安装的 PHP 也无所谓。只要本小节之前的那个测试过程能够完全通过，就可以进行开发了。

当然这只是我个人的看法，如果你觉得这个小工具能够帮到你的忙，那么你尽管用就可以了，不用在乎我的看法。

将扩展项目导入 Netbeans

下面，我们忽略上一节所讲的所有内容。接下来的内容是紧接上一节之前的内容的。

要编辑扩展中的每个文件，用 vi 当然没问题，但对我这个懒人来说，用 Netbeans 更顺手一些。所以这里介绍一下如何把下载的扩展源文件导入到 Netbeans 中去。

导入其实很简单，打开 Netbeans 的菜单，选择 [文件] > [新建项目]，然后选择类别为 C/C++，项目为基于现有源代码的 C/C++ 项目，如图所示：

然后点击浏览按钮，选择我们通过 git 下载的那个文件夹：

最后点完成就可以了。

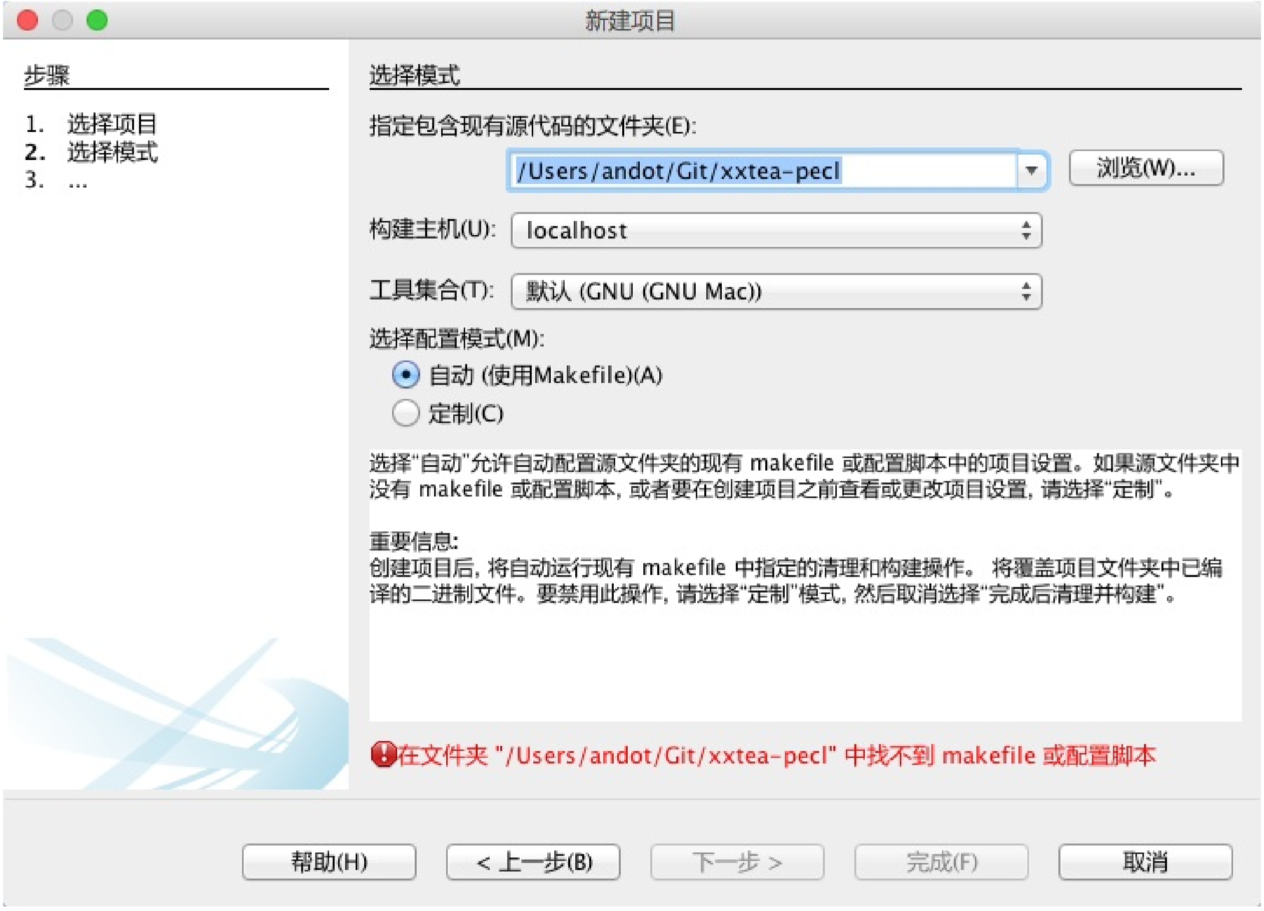
如果你只是用 `git clone` 命令下载了那个项目，而没有执行：

```
phpize
```

或者执行了

```
phpize --clean
```

清除了生成的所有文件，那么你在刚才的界面里选择文件夹后，会看到这样的错误提示：



不用着急。重新执行一下 `phpize`，然后再选择目录就可以了。

PHP 扩展的头文件

PHP 扩展至少有一个头文件，它的名字为 php_extname.h，其中 extname 为 PHP 扩展的名称。例如：在 xxtea 扩展中，这个文件就是 php_xxtea.h。

因为 xxtea 是一个小扩展，所以这个头文件本身就很小。如果我们要创建一个比较大的扩展，比如其中包含了近百个函数或多个类，那么我们应该把这些函数或类的声明分散到其它的头文件中，让这个头文件保持尽可能的小，只包含有限的一组声明。库或系统的头文件不应该被包含在其中。它一般也就包含扩展模块函数的声明（例如 PHP_MINIT_FUNCTION），模块的入口指针以及版本号的定义。至于原因嘛，简单来说就是为了加快同 PHP 一起静态编译时的速度，减少可能发生冲突的机会。

xxtea 扩展的头文件很简单，全部代码如下：

```
/**********************************************************\
|                                                          |
| php_xxtea.h                                              |
|                                                          |
| XXTEA for pecl include file.                             |
|                                                          |
| Encryption Algorithm Authors:                            |
|      David J. Wheeler                                    |
|      Roger M. Needham                                    |
|                                                          |
| Code Author:  Ma Bingyao <mabingyao@gmail.com>           |
| LastModified: Apr 06, 2015                               |
|                                                          |
\**********************************************************/

#ifndef PHP_XXTEA_H
#define PHP_XXTEA_H

#ifdef HAVE_CONFIG_H
#include "config.h"
#endif

#include "php.h"

extern zend_module_entry xxtea_module_entry;
#define phpext_xxtea_ptr &xxtea_module_entry

#define PHP_XXTEA_MODULE_NAME   "xxtea"
#define PHP_XXTEA_BUILD_DATE    __DATE__ " " __TIME__
#define PHP_XXTEA_VERSION       "1.0.11"
#define PHP_XXTEA_AUTHOR        "Ma Bingyao"
#define PHP_XXTEA_HOMEPAGE      "https://github.com/xxtea/xxtea-pecl"

ZEND_MINIT_FUNCTION(xxtea);
ZEND_MSHUTDOWN_FUNCTION(xxtea);
ZEND_MINFO_FUNCTION(xxtea);

/* declaration of functions to be exported */
ZEND_FUNCTION(xxtea_encrypt);
ZEND_FUNCTION(xxtea_decrypt);
ZEND_FUNCTION(xxtea_info);

#endif /* ifndef PHP_XXTEA_H */
```

其中开头的注释是版权声明，文件信息说明等等，你喜欢写什么就写什么，什么都不写也没关系。只有一点需要注意，那就是如果你要写注释，你需要用C语言风格的注释：

```
/* 我是 C 语言风格的注释 */
```

不要用C++风格的注释：

```
// 我是 C++ 风格的注释
```

不仅仅是开头的这段注释，整个扩展中所有的注释都需要写成C语言风格的注释。

原因参见 PECL/PHP 编码标准

> 1. Never use C++ style comments (i.e. // comment). Always use C-style comments instead. PHP is written in C, and is aimed at compiling under any ANSI-C compliant compiler. Even though many compilers accept C++-style comments in C code, you have to ensure that your code would compile with other compilers as well. The only exception to this rule is code that is Win32-specific, because the Win32 port is MS-Visual C++ specific, and this compiler is known to accept C++-style comments in C code.

翻译如下：

> 1. 绝不要使用 C++ 风格的注释（即 // 注释）。全用 C 风格的注释代替就对了。PHP 是用 C 写的，旨在任何 ANSI-C 兼容的编译器下都可编译。尽管许多编译器在 C 代码中接受 C++ 风格的注释，但是你要确保你的代码在其它的编译器上也能正常工作。该规则的唯一例外是为 Win32 编写的特定代码，因为 Win32 的移植版本是特定于 MS-Visual C++ 编译器的，而该编译器是明确可以在 C 代码中接受 C++ 风格注释的。

接下来的

```
#ifndef PHP_XXTEA_H
#define PHP_XXTEA_H
...
#endif /* ifndef PHP_XXTEA_H */
```

这几行是 C 语言头文件的基本写法，本来不想多做解释的，但考虑到有些同学原来只是做 PHP 的，对 C 语言基础也未必掌握，这里就啰嗦两句。

这几行的目的是为了保证该 .h 文件被多次包含的时候，防止它中间的代码被重复执行。

如果你还是不明白是什么意思，你还是先去看看 C 语言基础的书吧。虽然本书是一本 PHP 扩展开发入门的书，但是如果你一点 C 语言基础都没有，后面的内容看了也白看。

如果你觉得你的 C 语言基础没有问题，那可以继续往下看：

```
#ifdef HAVE_CONFIG_H
#include "config.h"
#endif

#include "php.h"
```

上面这几行其实也可以不放在这个头文件中，放到扩展的主文件开头是一样的[1]。不过那样的话，它后面的几行所使用的 `zend_module_entry` 这些类型，`ZEND_MINIT_FUNCTION` 这些宏，在这个头文件中就找不到定义了，在某些编辑器中打开可能会有错误提示。所以我把它放在了这个头文件中。反正这几行代码对静态编译不会有什么影响。

```
extern zend_module_entry xxtea_module_entry;
#define phpext_xxtea_ptr &xxtea_module_entry
```

这两行是重点，这两行声明的是模块的入口，照着写就行了。如果你用 ext_skel，它也会帮你生成这两行。

接下来这几行：

```
#define PHP_XXTEA_MODULE_NAME   "xxtea"
#define PHP_XXTEA_BUILD_DATE    __DATE__ " " __TIME__
#define PHP_XXTEA_VERSION       "1.0.11"
#define PHP_XXTEA_AUTHOR        "Ma Bingyao"
#define PHP_XXTEA_HOMEPAGE      "https://github.com/xxtea/xxtea-pecl"
```

只有

```
#define PHP_XXTEA_VERSION       "1.0.11"
```

是重点，其它的可有可无。

把跟扩展有关的常量宏都这样定义的好处是，后面的主文件写起来会比较清楚，也比较好改。所以建议这样做。

这里还有一点要注意，这些常量宏的格式必须是 `PHP_EXTNAME_XXX` 这样的风格。尤其是 `PHP_EXTNAME_VERSION`。

pecl 网站提供了一种版本检查的机制，如果你上传的扩展代码中的实际版本号和 package.xml 写的版本号不一致（比如你忘了改 php_extname.h 中的版本号，或者忘了改 package.xml 中的版本号），pecl 网站都会检查出来，给你一次改正的机会。

但如果你没有采用 `PHP_EXTNAME_VERSION` 这样风格的宏来定义版本号，那 pecl 网站的在你上传扩展时就检查不出来了。

我曾经就犯过这个错误，导致 pecl 网站上的 xxtea 扩展有几个版本的版本号实际上是不对的。但我也没机会再去修正那些错误的版本号了。这都是血和泪的教训啊，切记，切记！

```
ZEND_MINIT_FUNCTION(xxtea);
ZEND_MSHUTDOWN_FUNCTION(xxtea);
ZEND_MINFO_FUNCTION(xxtea);
```

这几行是关于模块的几个函数的声明。它们是可有可无的。你只要保证在后面的主文件中，引用这些函数名符号时，它们已经被声明，或已经被定义就行了。

```
/* declaration of functions to be exported */
ZEND_FUNCTION(xxtea_encrypt);
ZEND_FUNCTION(xxtea_decrypt);
ZEND_FUNCTION(xxtea_info);
```

这几行是关于导出到 PHP 中的函数的声明。同样，它们是可有可无的。原则跟上面说的一样。

你可以把它们从这里移动到扩展的主文件开头，或者在主文件中把它们的定义移到它们被引用之前。

简单来说，可以用一句话总结：

> 你程序中引用的每个符号，要么提前声明，要么提前定义，否则在编译或连接的时候，会出现找不到符号的错误。

而关于本节内容的总结呢，也是一句话：

> 在 php_extname.h 中，只有模块入口声明和模块版本号定义是必须的，其它的都是可有可无的。但只要不影响扩展正常工作，就让它保持尽可能的小。

---

[1] ext_skel 生成的头文件中就不包含这几行，而是放在了扩展的主文件中。

PHP 扩展的主文件